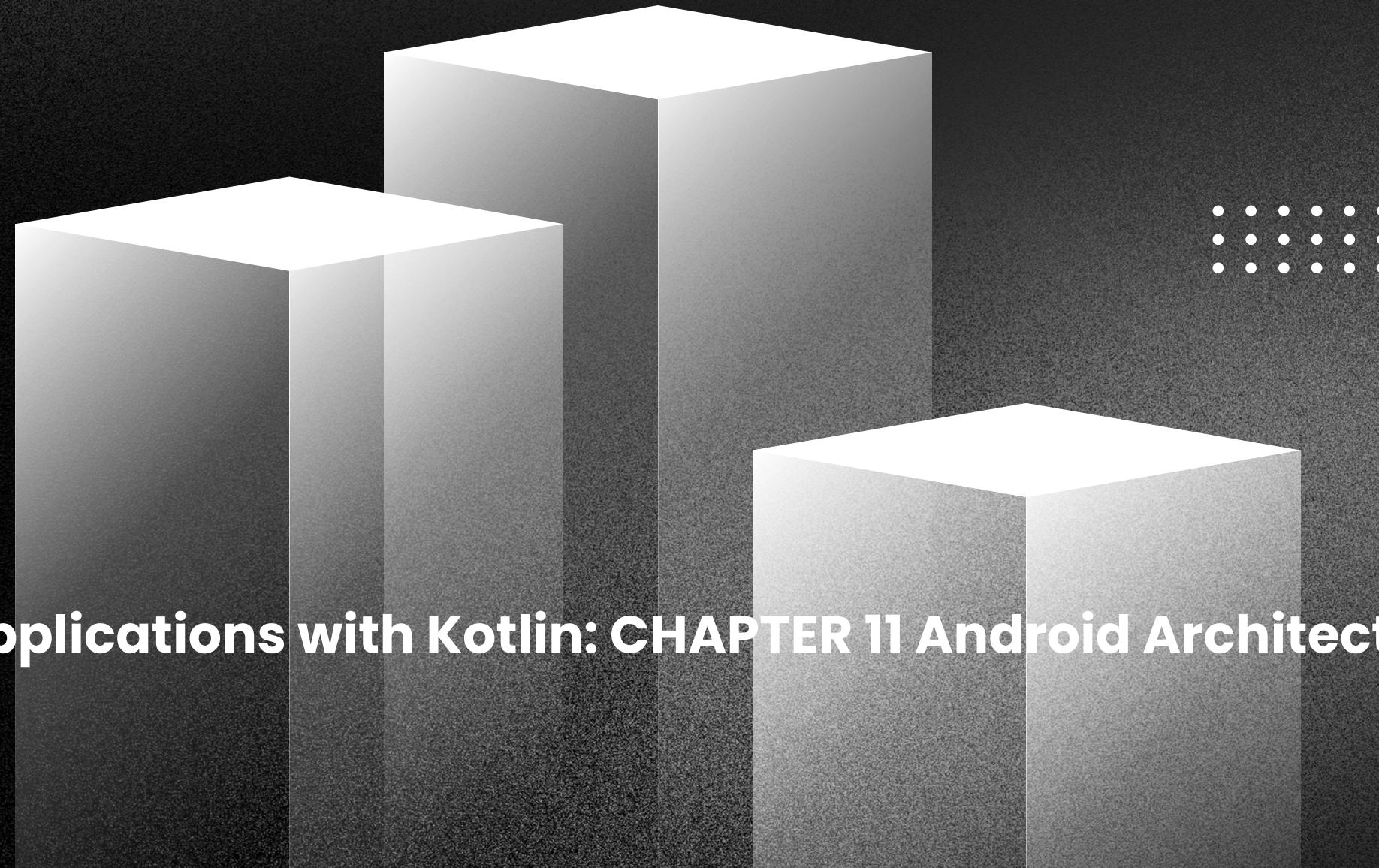


JETPACK COMPOSE & VIEWMODEL

Kiến trúc hiện đại cho Android linh hoạt, tối ưu và dễ bảo trì

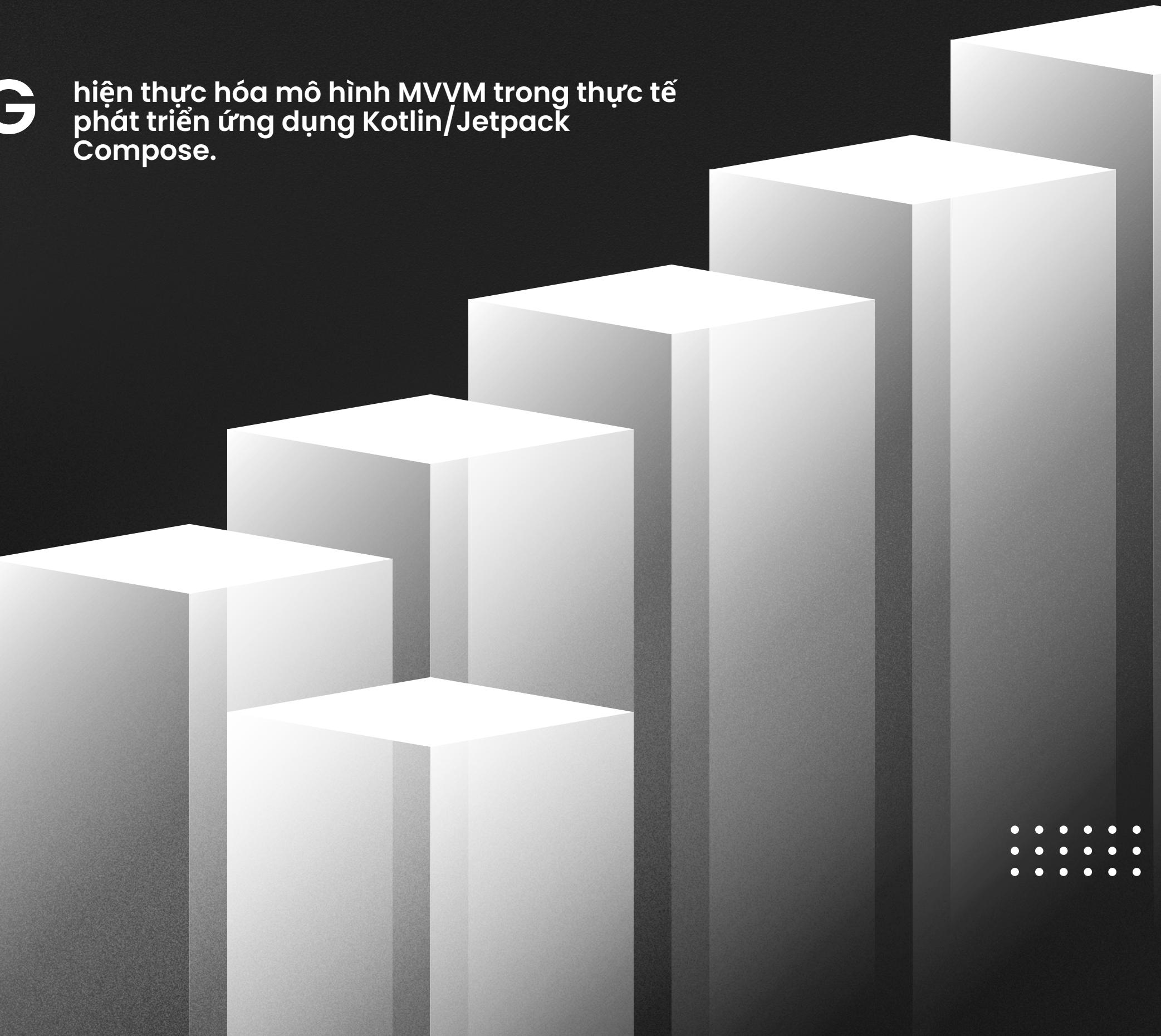


How to Build Android Applications with Kotlin: CHAPTER 11 Android Architecture Components

TỔNG QUAN NỘI DUNG

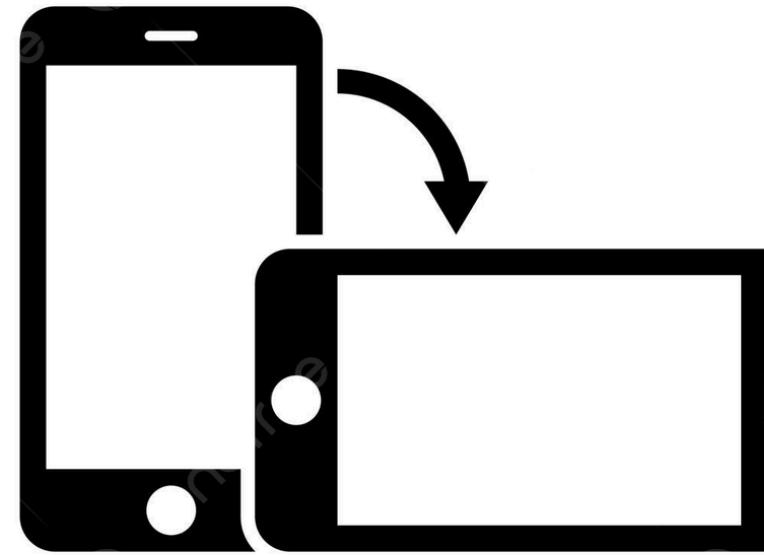
- Vấn đề thực tế
- Giới thiệu ViewModel
- Giới thiệu SavedStateHandle
- Giới thiệu Room
- Kết hợp ViewModel + Room
- Ví dụ minh họa tổng thể
- Ưu điểm và thực tiễn, MVC và MVVM cái này tốt ?
- Những lưu ý kỹ thuật
- Kết luận

hiện thực hóa mô hình MVVM trong thực tế
phát triển ứng dụng Kotlin/Jetpack
Compose.



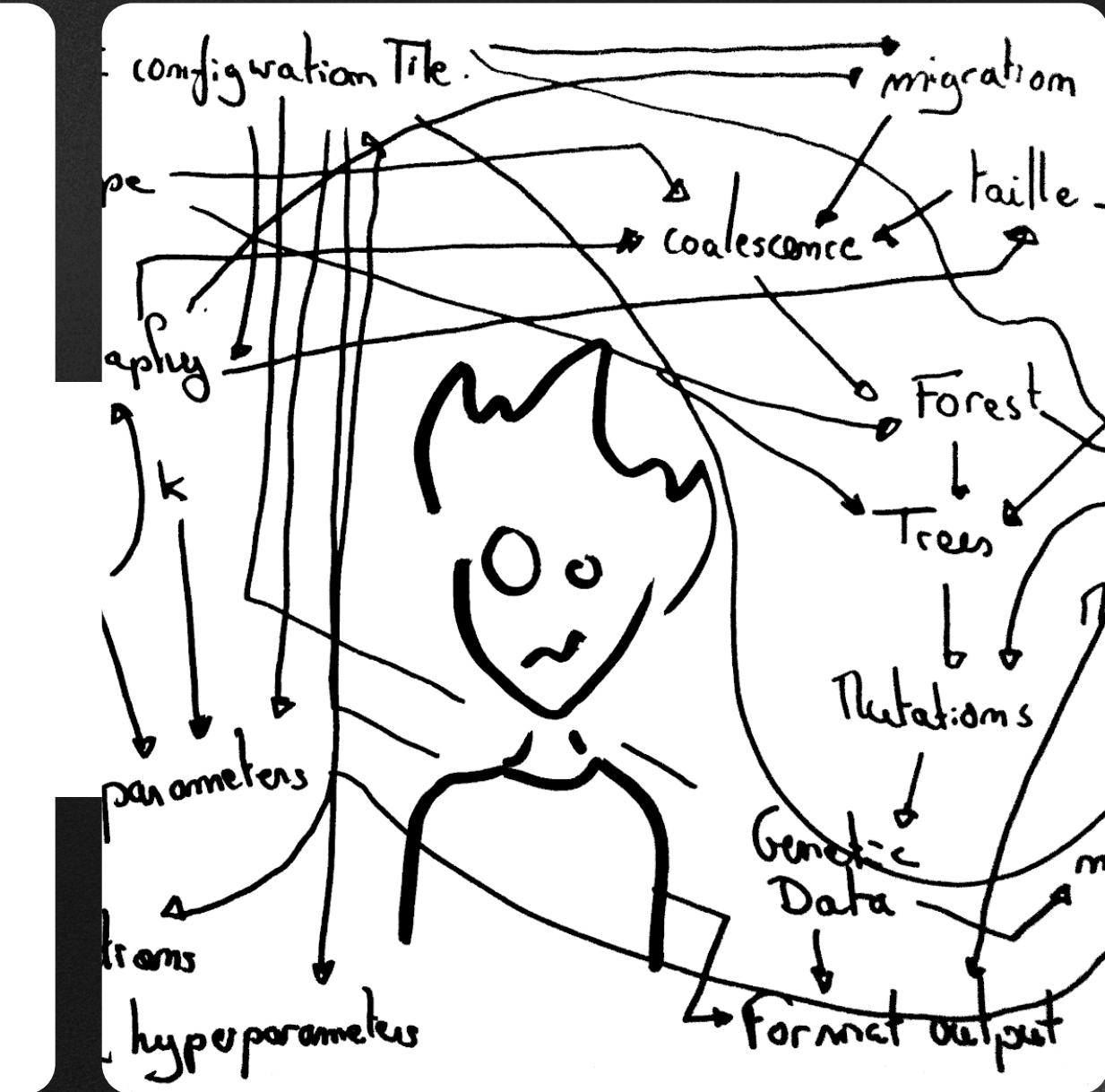
Vấn đề thực tế

Vấn đề khi không có Architecture Components



Dữ liệu mất khi xoay màn hình.

Nguyên nhân là do khi xoay màn hình, Activity hiện tại sẽ bị hủy và tạo lại. Khi Activity bị hủy, các dữ liệu được lưu trữ trong Activity sẽ bị mất.



Code UI và logic trộn lẫn.

Thói quen nhồi nhét mọi xử lý vào một nơi (thường là file code-behind, Activity hay View controller), dẫn đến code dài, khó debug và khó tái sử dụng.



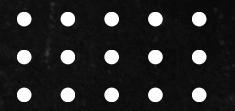
Khó kiểm thử và mở rộng.

- Khi code không phân lớp rõ ràng, muốn thêm hoặc thay đổi một chức năng phải sửa nhiều chỗ cùng lúc, dễ gây lỗi dây chuyền.
- Các thành phần phụ thuộc lẫn nhau, mỗi thay đổi sẽ ảnh hưởng tới toàn bộ hệ thống, code khó chăm sóc lâu dài và khó phát triển thêm module mới.



Giới thiệu ViewModel

ViewModel – Giữ dữ liệu sống qua vòng đời UI



📌 Nội dung cần nhớ:

1. Lưu và quản lý dữ liệu cho UI.
2. Sống lâu hơn Activity/Fragment.
3. Không bị reset khi xoay màn hình.

💡 VD minh họa :

- Model: Nơi lưu dữ liệu.
- View: Giao diện người dùng, chỉ hiển thị và nhận tương tác.
- ViewModel: Nơi xử lý logic và quản lý dữ liệu, giữ cho dữ liệu luôn mới.

Ví dụ như bạn muốn đếm số lần bấm nút:

- ViewModel sẽ giữ số đếm và tăng nó khi bạn bấm.
- View sẽ quan sát số đếm đó và tự động hiển thị lên màn hình.

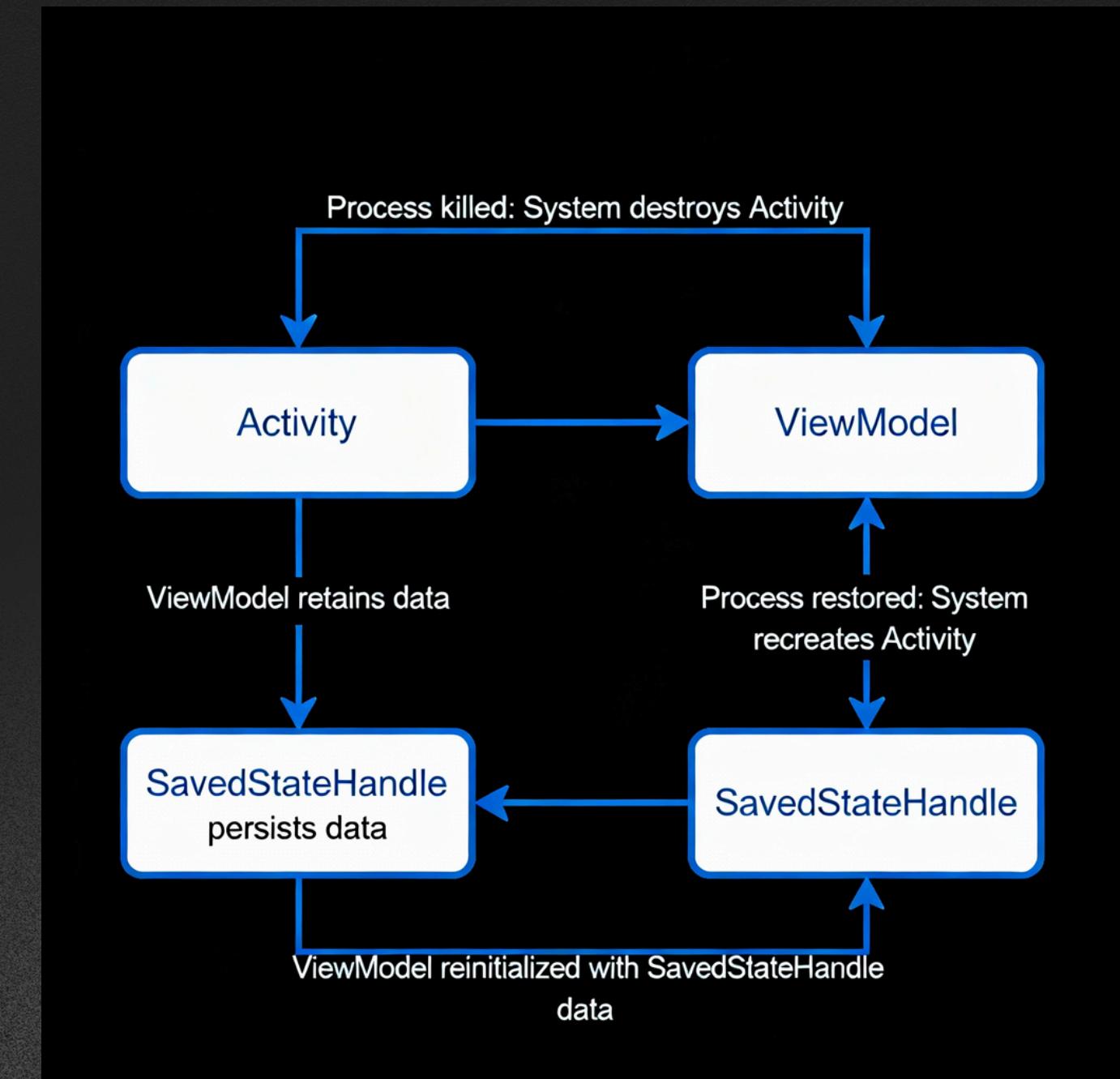
```
class MainViewModel : ViewModel() {  
    val count = MutableLiveData<Int>(0)  
    fun increase() { count.value = count.value?.plus(1) }  
}
```



Giới thiệu SavedStateHandle

SavedStateHandle – Khôi phục trạng thái bị kill process

- Lưu dữ liệu khi hệ thống thu hồi tài nguyên.
- Tự động khôi phục dữ liệu khi khởi động lại.





Giới thiệu Room

Room – Lưu trữ dữ liệu bền vững với SQLite

```
@Entity data class User(@PrimaryKey val id: Int, val name: String)  
@Dao interface UserDao { @Query("SELECT * FROM User") fun getAll(): List<User> }
```

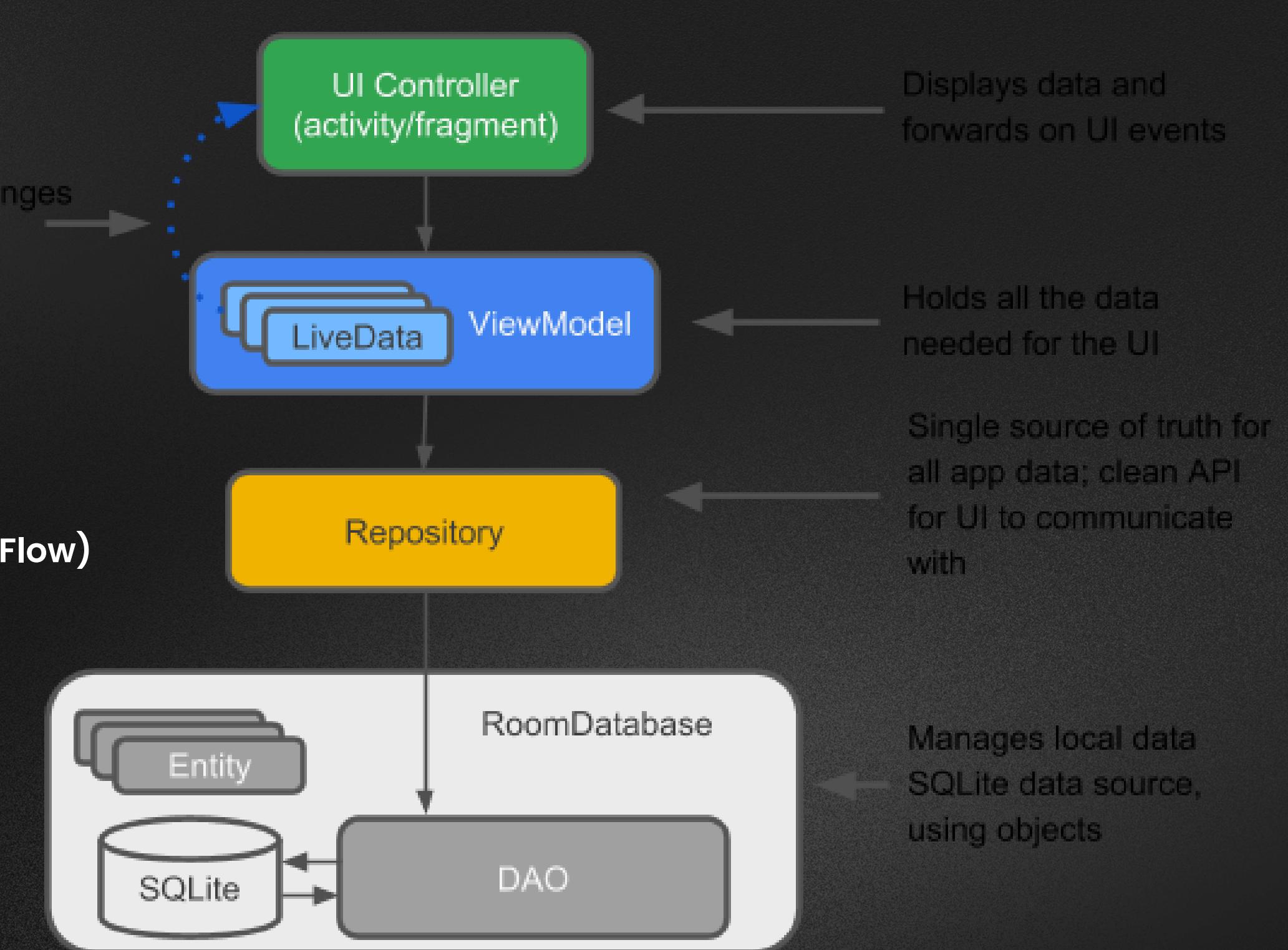
- Entity → Bảng.
-
- DAO → Truy vấn dữ liệu.
-
- Database → Kết hợp các DAO.

Kết hợp ViewModel + Room

Kết nối thành kiến trúc chuẩn Android

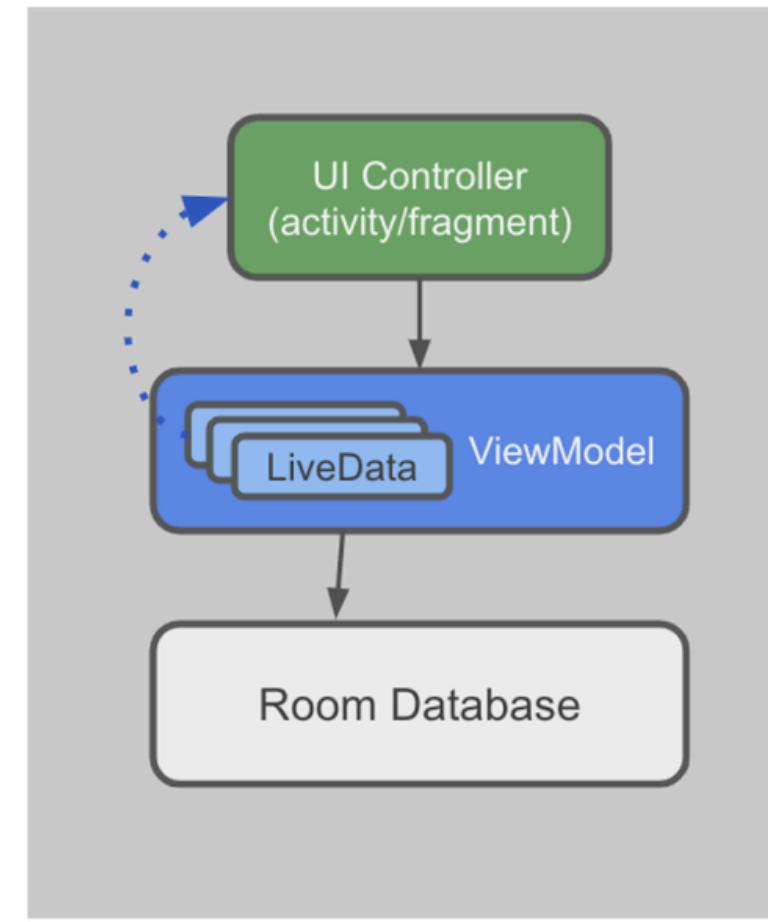
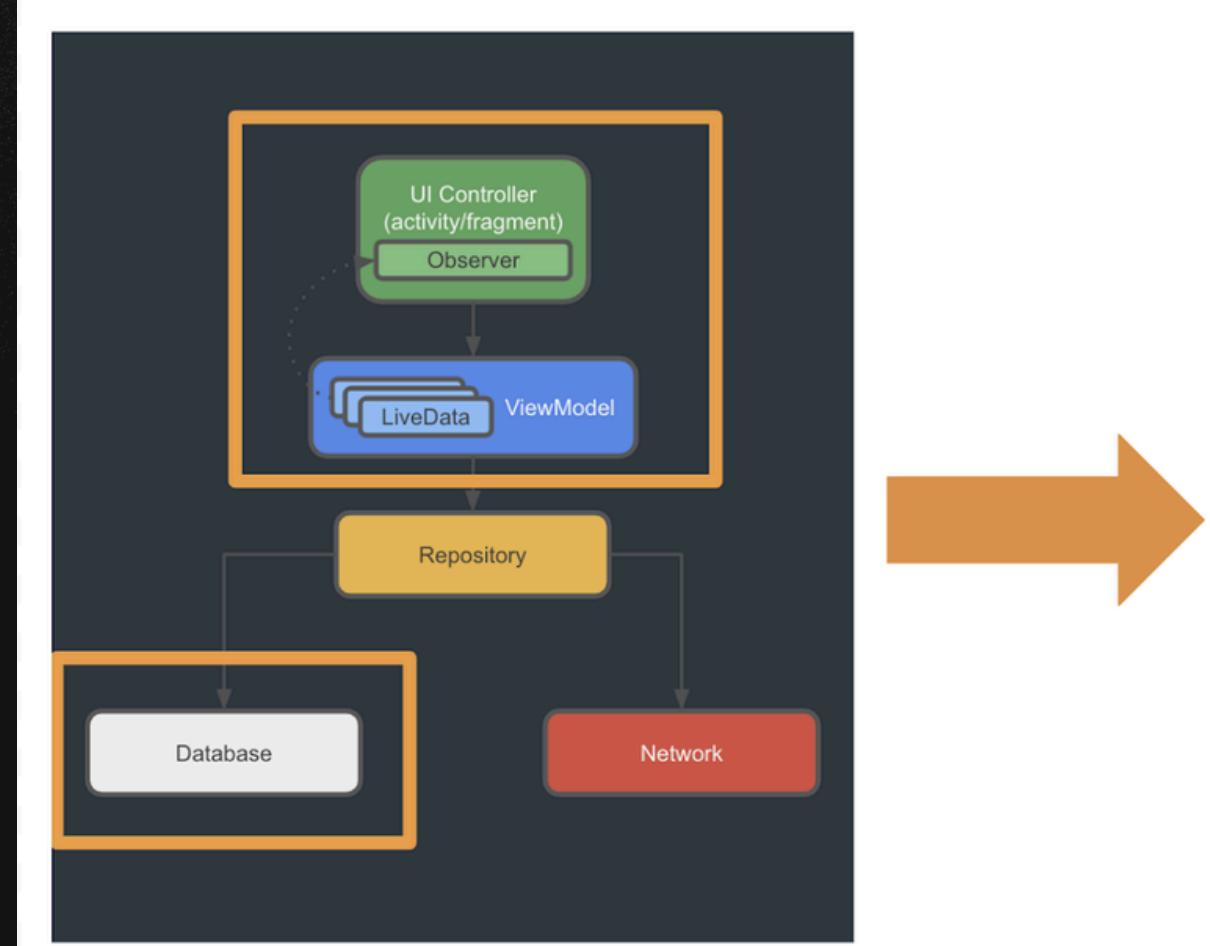
- UI → ViewModel → Repository → Room
- Tách biệt trách nhiệm rõ ràng
- Dữ liệu luân chuyển theo dòng quan sát (LiveData/Flow)

UI is notified of changes
using observation



Ví dụ minh họa tổng thể

Luồng dữ liệu trong ứng dụng ToDo list



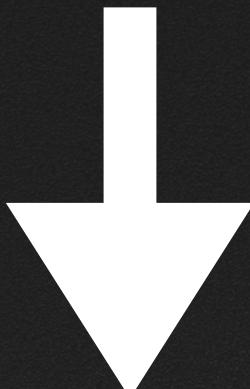
Sơ đồ ví dụ

Compose UI

ViewModel

Repository

Room Database



Ưu điểm và thực tiễn MVC và MVVM cái nào tốt khi làm ứng dụng?



Tại sao nên áp dụng Architecture Components

- Giảm rủi ro mất dữ liệu.
- Tăng khả năng tái sử dụng.
- Dễ kiểm thử, dễ bảo trì.
- Chuẩn hóa theo hướng dẫn Jetpack.

MVC (Model – View – Controller)

- **Model:** Quản lý dữ liệu, logic nghiệp vụ.
- **View:** Giao diện hiển thị.
- **Controller:** Trung gian xử lý sự kiện từ View và cập nhật Model.
- **X** *Vấn đề:* Controller thường bị phình to (God Object), khó bảo trì khi logic UI phức tạp.

MVVM (Model – View – ViewModel)

- **Model:** Dữ liệu, repository, API.
- **ViewModel:** Xử lý logic hiển thị, lưu trạng thái với `SavedStateHandle` và `LiveData / StateFlow`.
- **View (Compose UI):** Quan sát ViewModel, chỉ phản ứng khi dữ liệu thay đổi.
- **✓ Ưu điểm:**
 - Giảm phụ thuộc giữa View và logic xử lý.
 - Dễ kiểm thử (ViewModel không phụ thuộc Android Context).
 - Hỗ trợ tốt cho Compose và LiveData/Flow.

- Chỉ nên sử dụng với các ứng dụng web truyền thống và API
- Dự án cần sự linh hoạt

- Ứng dụng có giao diện người dùng phức tạp
- Dự án cần khả năng kiểm thử cao
- Dự án sử dụng liên kết dữ liệu

Những lưu ý kỹ thuật

Best Practices

- Không để ViewModel phụ thuộc trực tiếp vào Context.
- Sử dụng coroutine hoặc Flow cho tác vụ bất đồng bộ.
- Xử lý Room trên background thread.
- Tách logic Repository rõ ràng.



Kết luận



- ViewModel quản lý trạng thái UI.
- Room cung cấp lưu trữ dữ liệu bền vững.
- Kết hợp cả hai → Kiến trúc hiện đại, ổn định, dễ mở rộng.



THANK FOR WATCHING

CẢM ƠN BẠN ĐÃ NGHE BÀI THUYẾT TRÌNH CỦA TÔI, CHÚC BẠN CÓ 1 NGÀY TỐT LÀNH